PLATAFORMA DE INTERACCIÓN VOCAL DE PROPÓSITO GENERAL BASADA EN SIP Y VXML

Daniel Bolaños, Fernando López, Javier Garrido, José Colás

Departamento de Ingeniería Informática, EPS Universidad Autónoma de Madrid
Carretera de Colmenar Viejo, km 15. 28049 Madrid, España
daniel.bolanos@uam.es, fj.lopez@uam.es, javier.garrido@uam.es, jose.colas@uam.es

RESUMEN

A continuación describimos nuestra plataforma vocal, que ha sido desarrollada por el HCTLab con el objetivo de posibilitar la creación de aplicaciones de respuesta automática para la interacción con el usuario a través de la voz, ya sea por medio de un teléfono convencional, un teléfono IP, o bien a través de cualquier agente de usuario SIP (Session Initiation Protocol). Esta plataforma está basada en SIP, VoiceXML (lenguaje de marcado que permite la creación de interfaces vocales con el usuario) y CCXML (lenguaje de marcado para la creación de aplicaciones de control de llamada avanzado) así como otros estándares relacionados con interacción hombre máquina a través de la voz propuestos por la W3C. También describimos la forma en la que se han seleccionado e integrado los componentes software, todos ellos de código abierto, a partir de los cuales se ha construido, así como los módulos de los que consta y los flujos de datos y control que intercambian entre sí en los diferentes escenarios de ejecución. Finalmente hablaremos de aspectos de usabilidad, rendimiento y utilidad de la plataforma como tecnología base para nuestra línea de investigación actual.

PALABRAS CLAVE

VoiceXML, CCXML, SIP, MRCP, Voice Platform, Internet Telephony.

1. INTRODUCCIÓN

Hoy en día muchas personas están familiarizadas con las aplicaciones vocales tradicionales de respuesta automática (IVR), como son por ejemplo las aplicaciones de atención al cliente o banca telefónica. Estas aplicaciones hacen uso de recursos como son el reconocimiento de voz y la síntesis de voz para obtener información del usuario y proporcionarle la información requerida respectivamente. VoiceXML [1] es un lenguaje basado en XML desarrollado por la W3C para la creación de diálogos vocales que hacen uso de recursos de reconocimiento de voz, síntesis de voz, reconocimiento DTMF, audio digital y grabación de audio en el contexto de aplicaciones telefónicas o interacción a través de la voz en general. Otra característica muy interesante de VoiceXML es que proporciona los mecanismos para la creación de diálogos de iniciativa mixta, que hacen posible una forma de interacción con el usuario mucho más rica. La gran ventaja en el uso de VoiceXML es que se ha convertido en el Standard más utilizado para el diseño de aplicaciones vocales, proporcionando una arquitectura abierta para la programación de este tipo de aplicaciones frente a los antiguos sistemas IVR cerrados o propietarios. Uno de las causas del éxito de VoiceXML es el hecho de haber llevado las ventajas del desarrollo WEB y el acceso a contenidos WEB al desarrollo de aplicaciones IVR. De esta forma desarrollar una aplicación Vocal con VoiceXML se convierte en algo muy similar y casi análogo a desarrollar un formulario WEB utilizando HTML.

La especificación 2.0 de VXML proporciona algunas primitivas básicas para el control de llamada a través de los elementos <disconnect> y <transfer>, sin embargo estos elementos no son suficientes para realizar un control de llamada adecuado. Estas carencias son especialmente notorias, como veremos, si lo que queremos es construir aplicaciones de control avanzado de llamada, como son aquellas que hacen uso de recursos de audioconferencia. Por tanto ha sido necesario utilizar un lenguaje específico de control de llamada para permitir el manejo de sesiones VXML asociadas a aplicaciones vocales. El lenguaje elegido es

CCXML (Call Control eXtensible Markup Language) [2] que ha sido especificado por la W3C para proporcionar control de llamada avanzado en sistemas de diálogo y plataformas de IVR. CCXML no fue creado exclusivamente para ser utilizado en conjunción con VXML sin embargo son dos especificaciones que están muy ligadas y permiten, como veremos, una integración prácticamente sin fisuras.

VoIP (voz sobre IP) engloba multitud de tecnología y protocolos de señalización y transporte destinados a hacer posible el establecimiento de llamadas y la transmisión de voz a través de redes de datos. Nosotros hemos elegido SIP como protocolo de señalización debido a que se ha convertido en el estándar de facto en VoIP, mientras que el protocolo de transporte utilizado es RTP. SIP [3] es un protocolo de señalización muy utilizado para el establecimiento y finalización de sesiones multimedia a través de Internet. SIP proporciona un mecanismo general de acceso a la plataforma desde dispositivos de muy diferente tipo, básicamente cualquier dispositivo que disponga de un agente de usuario SIP puede interaccionar con el sistema, por ejemplo teléfonos móviles IP, teléfonos SIP software e incluso teléfonos convencionales a través de un switch PSTN-SIP.

2. ARQUITECTURA DE LA PLATAFORMA

La arquitectura de la plataforma es completamente distribuida de modo que todos los módulos exceptuando el MediaServer pueden ser replicados incluso dentro del contexto de ejecución de una misma aplicación vocal. Esto es fundamental cuando se diseñan aplicaciones vocales que acceden a módulos de reconocimiento automático de voz de habla continua, que generalmente tienen un elevado consumo de CPU.

A. Características generales.

1. Desarrollada 100% a partir de componentes software de código abierto.
2. Independiente de la plataforma: permite trabajar en entorno Linux y Windows.
3. Todos sus módulos están desarrollados en C++ utilizando STL a excepción del parser de gramáticas DTMF en formato BNF, que está desarrollado en Java.
4. Completamente distribuida: todos los módulos tienen contextos de ejecución diferentes y pueden situarse en máquinas diferentes.
5. Escalabilidad: todos los módulos a excepción del MediaServer se pueden replicar.
6. Grado de estandarización: incorpora los últimos estándares en comunicación mediante VoIP así como diseño de aplicaciones vocales y control avanzado de llamada:
    a. Especificación VXML 2.1 en el intérprete de diálogos vocales.
    b. Especificación CCXML 1.0 para el soporte de operaciones de control avanzado de llamada.
    c. Protocolo de señalización SIP, RFC 3261, en combinación con SDP, RFC2327.
    d. Protocolo de transporte RTP, RFC1889.
    e. Protocolo de control de recursos multimedia MRCP, RFC 4463, para la comunicación con servidores de voz en tiempo real.
7. Interoperabilidad con multitud de dispositivos, desde teléfonos móviles SIP hasta teléfonos convencionales a través de la RTC (Red Telefónica Conmutada) por medio de un gateway SIP/RTC.

B. Módulos que la componen (ver Figura 1).

MediaServer: se trata del módulo central de la plataforma, su funcionalidad es la siguiente:

1. Se encarga de actuar de agente de usuario SIP para cada una de las conexiones que se establecen con agentes de usuario en dispositivos externos, por ejemplo teléfonos IP. De esta forma mantiene un bucle de eventos de señalización que en combinación con los comandos de control de llamada que recibe del intérprete de documentos CCXML le permite llevar a cabo el control de llamada. Para llevar a cabo esta funcionalidad dentro del MediaServer se ha utilizado la librería de código abierto sipXtapi perteneciente

a SIPFoundry [4], que es una comunidad de desarrollo de software libre creada para fomentar la adopción de SIP. SipXtapi proporciona una pila SIP compatible con la RFC3261 así como generación de tonos DTMF fuera de banda según la RFC2833, reproducción y grabación de audio digital en diferentes formatos, transferencia de llamada, audioconferencia y muchas otras características interesantes.

2. Inicializa y gestiona los recursos necesarios para la ejecución de las aplicaciones vocales, es decir, se encarga de gestionar las sesiones que sea necesario crear en el resto de módulos.
3. Proporciona un mecanismo de comunicación con aplicaciones de terceros que pretendan hacer uso de los recursos del MediaServer. Un caso típico son las aplicaciones que utilizan la plataforma para realizar llamadas de iniciativa saliente. Típicamente estas llamadas se realizan en respuesta a eventos que por ejemplo se producen en una base de datos al cambiar un registro, etc. En la Figura 1, podemos ver como las entidades AP. 1 AP. 2 y AP. 3 (se trata de aplicaciones externas a la plataforma creadas, por ejemplo, para generar avisos telefónicos en determinadas circunstancias como alarmas, etc) se comunican con el MediaServer a través de mensajes xml, el protocolo de comunicación es el siguiente:

   Ap. 1 → MediaServer: la aplicación envía un mensaje xml mediante TCP al MediaServer indicándole el *anis* (entidad SIP que debe aparecer en los mensajes SIP como remitente de la llamada), el *dnis* (entidad SIP destinataria de la llamada) y el appname (identificador de la aplicación, este campo lo necesita el MediaServer para localizar dentro de la denominada tabla de enrutado el documento CCXML raíz que deberá ser ejecutado en relación con esta aplicación, este documento contendrá típicamente al menos un elemento del tipo <createcall> o <createconference>). Muchas veces los campos anis y dnis pueden aparecer vacíos, en este caso será el propio documento CCXML el que resuelva esta información al ser generado dinámicamente en un servidor WEB.

   ```
   <outcomingcall anis="" appname="Dedicatorias" dnis="sip:dani@mydomain"/>
   ```

   Ap. 1 ← MediaServer: el MediaServer envía una confirmación de la petición con el *sessionid* (se trata del identificador de la sesión de control de llamada que ha sido creada y que puede ser útil a la aplicación a modo de registro o para la recepción de posteriores mensajes relativos a esta sesión).

   ```
   <outcomingcall sessionid="session 2" status="outcomingcall.status.success"/>
   ```

4. Almacenamiento y obtención del servidor WEB de Aplicación, de audio digitalizado y otros recursos multimedia que sean necesarios durante la interacción con el usuario en el contexto de ejecución de las aplicaciones vocales.
5. Comunicación directa con motores de síntesis y reconocimiento de voz a través del protocolo MRCP [5]. MRCP (Media Resource Control Protocol) es un protocolo de comunicaciones creado por la IETF que permite a los servidores de voz proporcionar servicios como el reconocimiento de voz o la síntesis de voz a sus clientes, en este caso agentes de usuario SIP. MRCP se basa en mensajes y respuestas siguiendo la filosofía de HTTP, sin embargo no proporciona un protocolo de transporte por lo que lo utilizaremos en colaboración con RTP para el envío del audio. MRCP es de gran utilidad cuando se quiere realizar reconocimiento o síntesis de voz en tiempo real, y es soportado en su versión 1.0 por la mayoría de los motores comerciales.
6. Balanceo de carga de CPU entre los diferentes módulos: existe un documento XML donde reside la información de los módulos que están disponibles dentro de la plataforma en un momento dado. Para cada módulo existe información de localización así como un atributo llamado “alias” que permite agrupar los módulos en conjuntos sobre los cuales se realizará el balanceo de carga. De esta forma cada vez que añadimos una nueva aplicación vocal en el sistema, tarea que se realiza a través de la tabla de enrutado, indicaremos el alias de los módulos que utiliza, y será el MediaServer el que, en tiempo de ejecución, decida el módulo que atenderá una determinada petición de entre todos aquellos que tengan el mismo alias. Para realizar esta tarea el MediaServer mantiene en memoria una tabla con las peticiones que está sirviendo en un momento dado cada módulo para así estimar cuál de todos ellos está en mejor disposición de atender una nueva petición en función de la carga de CPU.

   ```
   <?xml version="1.0" ?>
   <Modules>
      <Module alias="htk_asr" type="ASR" ip="127.0.0.1" ... />
      <Module alias="scansoft_tts" type="TTS" ip="127.0.0.1" ... />
   ```

```
    <Module alias="vxi" type="VXML" ip="127.0.0.1" ... />
    <Module alias="cxi" type="CCXML" ip="127.0.0.1" ... />
    <Module alias="mediaserver" type="CTI" ip="127.0.0.1" ... />
</Modules>
```

Gestor de diálogos: este módulo se encarga de gestionar la ejecución de diálogos vocales VXML una vez que una conexión entre dos o más agentes de usuario se ha establecido, para ello utiliza el intérprete VXML de código abierto OpenVXI [6] en su versión 3.0, que satisface la especificación VoiceXML 2.0 prácticamente al 100%. Este módulo básicamente recibe comandos de diálogo del MediaServer a los que responde emitiendo eventos de diálogo en función de las transiciones definidas en el documento CCXML que se esté procesando. Durante la ejecución de los diálogos se encarga de realizar peticiones de recursos telefónicos tanto al MediaServer como a los módulos de reconocimiento y síntesis de voz dependiendo de la aplicación vocal. Estas peticiones son típicamente grabar audio, reproducir audio, reconocer (con una gramática asociada) voz, reconocer DTMF y sintetizar voz.

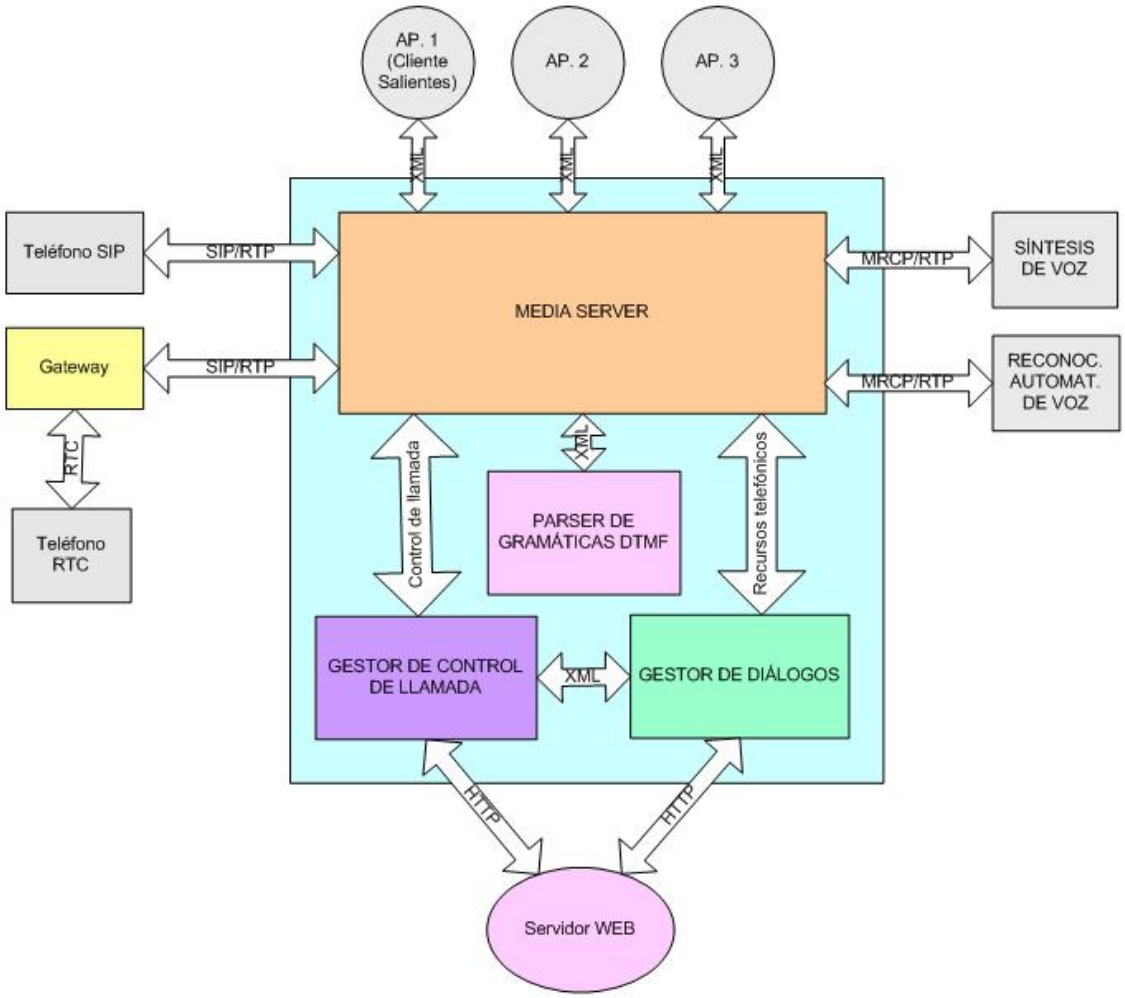

Figura 1: Arquitectura general de la plataforma e interacción con dispositivos/entidades externas.

Gestor de control de llamada: este módulo se encarga de gestionar la ejecución de documentos de control de llamada CCXML 1.0 asociados a aplicaciones vocales (ver Tabla de Enrutado). El intérprete CCXML que utiliza para procesar los documentos ha sido construido a partir de librerías que forman parte del proyecto Oktopous PIK [7], que es una iniciativa de código abierto alojada en SourceForge [8] por Phonologies [9]. La funcionalidad de este módulo consiste en gestionar la ejecución de sesiones de procesado de documentos CCXML que obtiene del Servidor WEB de Aplicación cada vez que se crea una nueva sesión, por ejemplo al recibirse una llamada. Recordemos que la notificación de nuevas llamadas y por tanto creación de sesiones la realiza el MediaServer, en el caso de llamadas entrantes mediante el evento connection.alerting y en el de salientes mediante un mensaje XML con la información necesaria. Este módulo consta fundamentalmente de dos hilos de ejecución, uno se encarga de recibir eventos de conexión, conferencia o diálogo provenientes del MediaServer y almacenarlos en una cola, mientras que el otro va sacando de la cola los eventos y envía los comandos de conexión, conferencia o diálogo al MediaServer según se vayan generando en función de las transiciones definidas en el documento CCXML que se esté procesando (ver Figura 2).

Parser de gramáticas DTMF: este módulo se encarga de procesar gramáticas de DTMF en formato BNF, estas gramáticas pueden encontrarse embebidas dentro del propio código de un documento VXML (entre etiquetas <grammar>) o bien aparecer como referencias externas, en este caso el parser obtendrá el documento con la gramática del Servidor WEB de Aplicación. La utilización de reconocimiento DTMF es de gran utilidad en aplicaciones en las que el usuario ha de elegir una opción entre varias en un menú (mediante los elementos <menu> y <choice> de VXML) o simplemente ha de rellenar un campo dentro de un formulario (mediante los elementos <field> y <option>). Para construir este módulo hemos utilizado las librerías de código abierto que forman parte del proyecto BNF for Java [10]. Este módulo básicamente realiza dos operaciones, por un lado permite la carga de nuevas gramáticas y por otro lado permite conocer si una secuencia de dígitos DTMF es válida para una gramática dada. El control de los timeouts que la especificación VXML establece para el reconocimiento DTMF, como son interdigittimeout, termtimeout y termchar se realiza externamente en el MediaServer.

Servidor WEB de Aplicación: se trata de un servidor WEB convencional, en el que se alojan todos los elementos necesarios para la ejecución de una aplicación vocal. Estos elementos son documentos VXML y CCXML, programación de lado servidor como ASP, JSP o Java Servlets para acceso a bases de datos o bien generación dinámica de los diálogos vocales o de control de llamada, ficheros de audio digitalizado, gramáticas de voz y DTMF, etc. El servidor WEB de aplicación también desempeña un papel importante cuando se utilizan motores de síntesis y reconocimiento basados en fichero. En estos casos el ServidorWEB permite el intercambio y almacenamiento de los archivos temporales necesarios para realizar estas operaciones, es decir, permite almacenar los archivos de voz grabada que servirán de entrada al reconocedor, así como los archivos de voz sintética generados por los motores de síntesis.

3. ESQUEMA DE INTERACCIÓN ENTRE MÓDULOS

A continuación (ver Figura 2) vamos a ver un poco más en detalle los flujos de datos y de control que intercambian los módulos de la plataforma vocal. Así mismo veremos de qué forma se han integrado los diferentes componentes software

1. Eventos relacionados con el control de llamada según la especificación CCXML 1.0, es decir eventos relacionados con conexiones y conferencias, estos son: connection.alerting, connection.progressing, connection.connected, connection.disconnected, etc, salvo el evento connection.signal, del que no hacemos uso en nuestra implementación.
2. Comandos relacionados con el control de llamada según la especificación CCXML 1.0. es decir: <accept>, <redirect>, <reject>, <createcall>, etc. Estos comandos los envía el Gestor de Control de Llamada al MediaServer según se los va encontrando en las transiciones del documento CCXML asociado a la aplicación vocal que esté procesando.
3. Comandos de diálogo según la especificación CCXML 1.0, en este caso, puesto que no hay preparativos relativos a la creación de la sesión en el intérprete de diálogos ni finalización explicita de esa sesión, tan

sólo implementamos el comando <dialogstart>, que se encargará de crear una sesión del intérprete de diálogos en el Gestor de Diálogos para un documento VXML dado.

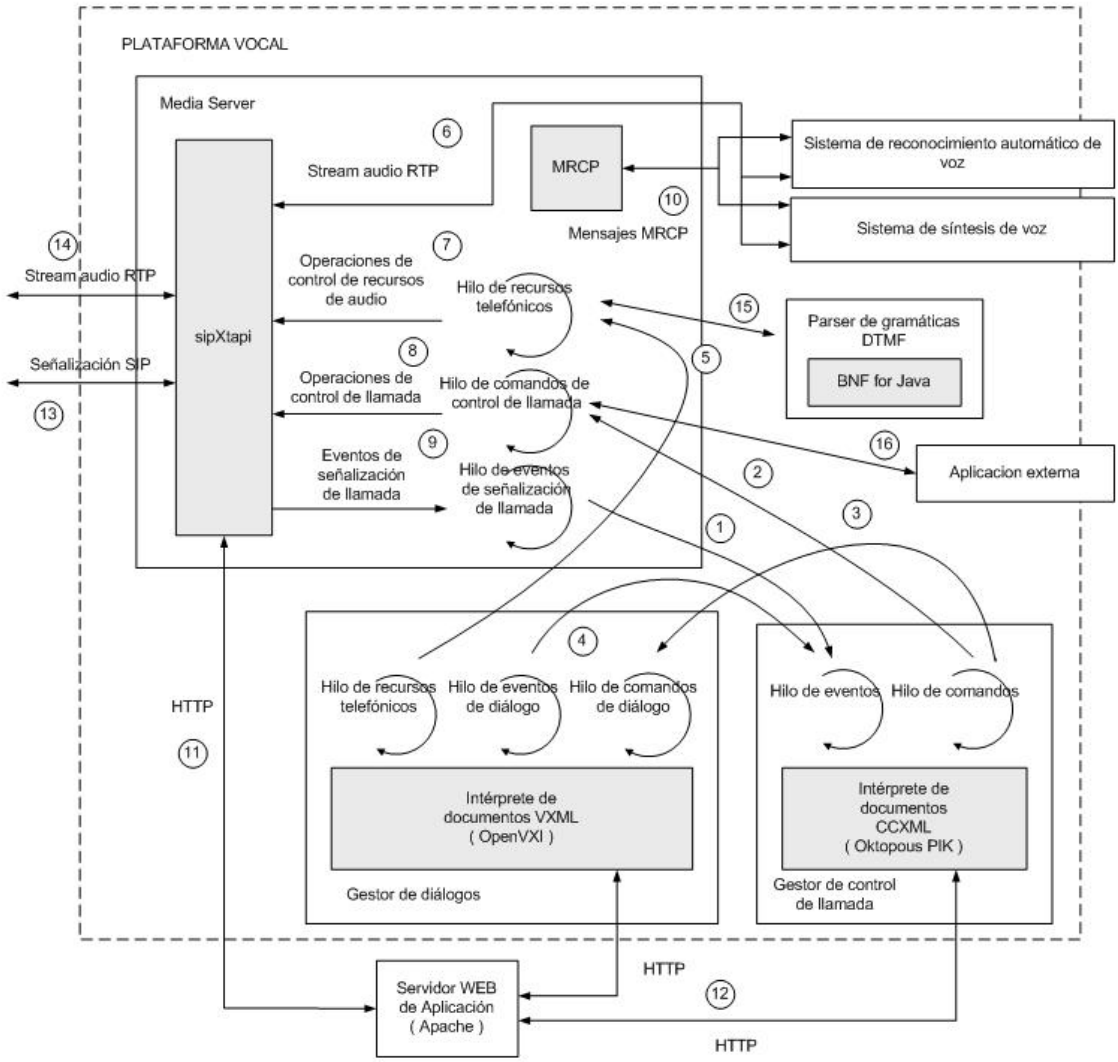

Figura 2: Esquema de interacción con los principales flujos de datos y control entre módulos.

4. Eventos de diálogo según la especificación CCXML 1.0. Se utilizan todos excepto los relativos a los preparativos para la creación de una sesión, es decir: dialog.prepared y error.dialog.notprepared.
5. Peticiones de recursos telefónicos, es decir: grabar audio digital, reproducir audio digital, conexión con servidores de voz, reconocimiento de DTMF, etc. Estas peticiones están típicamente relacionadas con los elementos <prompt>, <audio>, <grammar>, etc, dentro de un documento VXML.
6. Stream RTP de audio digital entre servidores de voz externos y el MediaServer, este stream es controlado por medio del uso del protocolo MRCP.
7. Utilización de las funciones de la librería sipXtapi relacionadas con los recursos telefónicos por parte del hilo de recursos telefónicos (se trata de funciones asíncronas).
8. Utilización de las funciones de la librería sipXtapi relacionadas con el control de llamada por parte del hilo de comandos de control de llamada (se trata de funciones asíncronas).
9. El hilo de eventos de señalización de llamada mantiene un bucle de eventos en el que recibe eventos de señalización de llamada desde la librearía sipXtapi, relativos a los cambios de estado de las conexiones y conferencias que hay establecidas.

10. El módulo cliente MRCP permite establecer comunicación con los servidores MRCP (servidores de voz) a través de mensajes y así hacer uso de los servicios que proporcionan.
11. La librería sipXtapi accede mediante HTTP a ficheros de audio digitalizado alojados dentro del Servidor WEB de Aplicación, por ejemplo cuando la aplicación vocal hace uso de locuciones pregrabadas.
12. Tanto el intérprete de documentos VXML como CCXML hacen uso de documentos alojados o generados por el Servidor WEB de Aplicación.
13. Intercambio de mensajes SIP entre los agentes de usuario de sipXtapi y los de dispositivos externos.
14. Stream RTP entre sipXtapi y los dispositivos externos, transporte de audio digitalizado.
15. Mensajes para el procesado de gramáticas DTMF, típicamente estos mensajes están relacionados con una primitiva <grammar> dentro de documentos VXML.
16. Mensajes para la comunicación entre el MediaServer y aplicaciones externas.

4. ASPECTOS DE RENDIMIENTO

La plataforma ha sido probada en condiciones reales para monitorizar el consumo de recursos, robustez, comportamiento frente a latencias en la red, así como aspectos de usabilidad relacionados con retardos, y timeouts. A continuación mostramos datos sobre el consumo de recursos en diferentes escenarios.

- Entorno hardware: se ha utilizado un Pentium IV a 2.4 GHz con 1GB de RAM. Se ha utilizado un gateway para la conexión con la plataforma a través de la RTC, se trata de un MP-104 FXO VoIP Gateway de AudioCodes.
- Entorno software: sistema operativo Windows XP sp2 así como todos los componentes necesarios para la instalación de la plataforma (todos los módulos han sido instalados en la misma máquina).

Hemos definido tres escenarios de prueba diferentes, en todos ellos se han lanzado 20 sesiones simultaneas en la plataforma y se han monitorizado los consumos de memoria y CPU, como se observa en la figura 3.

	Consumo memoria	Consumo CPU
Escenario 1*	471MB	89%
Escenario 2**	367MB	86%
Escenario 3***	635MB	100%

Figura 3: Consumo de memoria y tiempo de CPU en diferentes escenarios.

* Aplicación vocal tipo “secretaria virtual” de iniciativa entrante en la que se utiliza transferencia de llamada.
* Aplicación vocal tipo “dedicatorias” de iniciativa saliente y amplia cobertura de la especificación VXML.
* Aplicación tipo “sistema de obtención de información” de iniciativa entrante en cuyos diálogos hay un gran volumen de código JavaScript

En la Figura 4 mostramos la relación de consumo de memoria/cpu entre los diferentes módulos de la plataforma. Mientras que el mayor consumo de memoria corresponde al Gestor de diálogos debido al uso masivo de las librerías de JavaScript (SpiderMonkey) y procesado de XML (Xerces), el consumo de CPU es mayor en el MediaServer debido al manejo de streams de audio digitalizado en tiempo real.

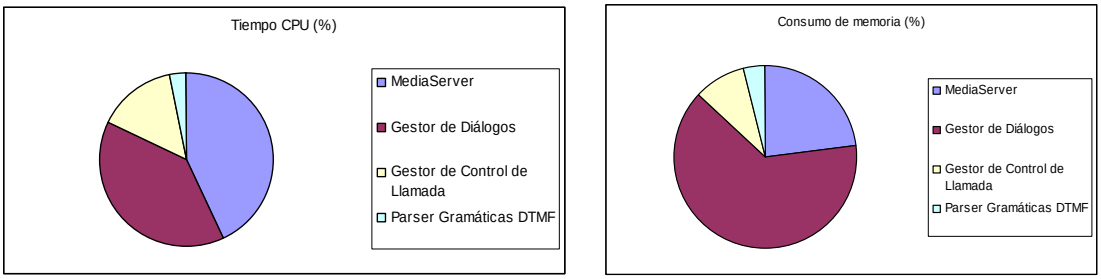

Figura 4: relación entre el consumo de memoria y tiempo de cpu de los distintos módulos.

5. CONCLUSIONES Y TRABAJO FUTURO

A día de hoy estamos trabajando en la puesta a punto de la plataforma para ser liberada como un proyecto de código abierto, como lo son todos los componentes a partir de los cuales se ha construido. Con este fin estamos realizando las pruebas necesarias en diferentes entornos y escenarios y reuniendo la documentación pertinente. El objetivo de esta iniciativa es proveer a la comunidad investigadora de una plataforma completamente estandarizada y abierta que facilite la experimentación en condiciones reales en el área de la interacción vocal y posteriormente también en el área de la interacción multimodal.

Dentro de nuestros intereses como grupo de investigación, el desarrollo de esta plataforma vocal nos permite disponer de la tecnología básica de interacción hombre-máquina a través de la voz que nos proporcione un punto de partida para nuestra experimentación en condiciones reales en aspectos como la medida de la satisfacción del usuario en relación a la interacción con aplicaciones basadas en diálogos de iniciativa mixta. Tarea a la que nos dedicamos actualmente.

Los diálogos de iniciativa mixta proporcionan una interacción con el usuario mucho más rica que los diálogos dirigidos, esto es porque hacen posible una forma de comunicación con el sistema muy similar a la forma en que se comunican las personas, es decir, el cambio de turno entre los componentes de la conversación se produce de una forma negociada mientras que en los diálogos de iniciativa dirigida uno de los componentes de la conversación es siempre el que dirige el cambio de turno. VoiceXML 2.0 permite la creación de diálogos de iniciativa mixta mediante el uso del elemento <initial> así como la definición de gramáticas a nivel de formulario (form-level grammars), a pesar de que también tiene sus limitaciones en este sentido, este es uno de los motivos por el que decidimos utilizar esta especificación. Por otro lado SIP nos proporciona un protocolo de acceso al sistema prácticamente desde cualquier dispositivo lo que es de gran importancia en la experimentación en el área de interacción multimodal.

Otro de los objetivos es la ampliación de la plataforma para permitir la interacción no sólo por voz sino también por video, obviamente el protocolo de señalización SIP en combinación con el protocolo de transporte RTP también permite realizar vídeo llamada, es más, SipXtapi ha incorporado recientemente esta funcionalidad dentro del agente de usuario. El video como medio de interacción resulta muy interesante en ocasiones en las que una comunicación oral no es posible, como es el caso de la interacción con las personas sordas, en este caso el sistema no sólo debe ser capaz de proveer información oral a través de voz sintética o audio digitalizado pregrabado sino que también deberá proporcionar video sintético con la información en lenguaje de signos para sordos, todo ello dentro de un contexto de interacción multimodal. En esta área de investigación están trabajando actualmente varios miembros del grupo HCTLab.

REFERENCIAS

[1] Especificación VoiceXML. http://www.w3.org/TR/voicexml20/

[2] Especificación CCXML. http://www.w3.org/TR/ccxml/

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[4] SIPFoundry, comunidad de desarrollo de software libre para fomentar la adopción de SIP. http://www.sipfoundry.org

[5] Media Resource Control Protocol (MRCP), RFC 4463

[6] OpenVXI. http://sourceforge.net/projects/openvxi/

[7] Oktopous PIK. http://sourceforge.net/projects/oktopous/

[8] Sourceforge, repositorio para desarrollo y almacenamiento de proyectos de código abierto. http://sourceforge.net/

[9] Phonologies. http://www.phonologies.com/

[10] BNF for Java, http://bnf-for-java.sourceforge.net

[11] K. Singh, A. Nambi, and H. Schulzrinne. Integrating VoiceXML with SIP services. In *Conference Record of the International Conference on Communications (ICC)*, May 2003